DIJKSTRA GRAPHS

Lucila S. Bento[1] Davidson Boccardo[2]

Raphael S. Machado[2] Flávio Miyazawa[3]

Vinicius G. Sá[1] Jayme L. Szwarcfiter[1,2]

[1] Universidade Federal do Rio de Janeiro

[2] Instituto Nacional de Metrologia, Qualidade e Tecnologia

[3] Universidade de Campinas

Computer programming before 1970

- Elegancy: not a main issue
- No general methodology
- No structure
- Main computer languages: machine code, assembler, FORTRAN, COBOL
- Main goal: Fit the program into memory

A computer program

Structured programming appears

- E. W. Dijkstra, Go to statements considered harmful (1968)
- E. W. Dijkstra, Notes on structured programming (1972)
- D. E. Knuth, Structured Programming with Go-To Statements (1974)

Structured programming spreads

- Kosaroju (1974), D-charts
- McCabe (1976), representation using graphs
- Appliying structured programming to the formulation of algorithms
  - Henderson and Snowdon (1972)
  - Knuth and Szwarcfiter (1974)

Structured programming takes over

With the time became a general technique for the formulation of algorithms

A natural question

- **How to tell if a given algorithm is structured ?**
- Apparently not yet considered.
- **How to tell if two structured algorithms have a similar structure (syntax) ?**
- Apparently not yet considered.

Our proposal

- Introduce a class of graphs which corresponds to structured algorithms, named **Dijkstra graphs**
- Answer the question of recognizing structured algorithms (i.e. Dijkstra graphs).
- Answer the question whether two structured algorithms are similar (i.e. solving the isomorphism problem for Dijkstra graphs).

The proposed algorithms for recognition and isomorphism of Dijkstra graphs have both linear time complexity.

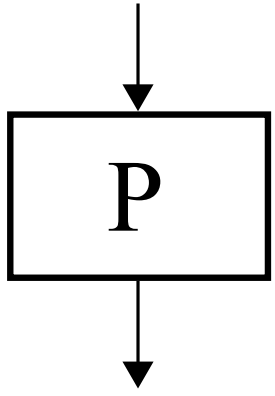
Contents

Basis of Structured Programming

E. W. Dijkstra, Notes on structured programming, in *Structured Programming*, Academic Press, 1-82, 1982.

Three types of statements:

Sequence:

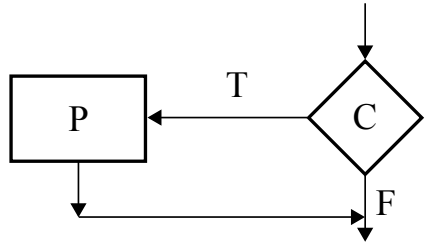


P

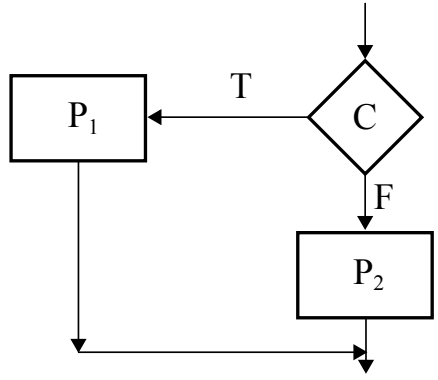
Basis of Structured Programming

Selection

SELECTION



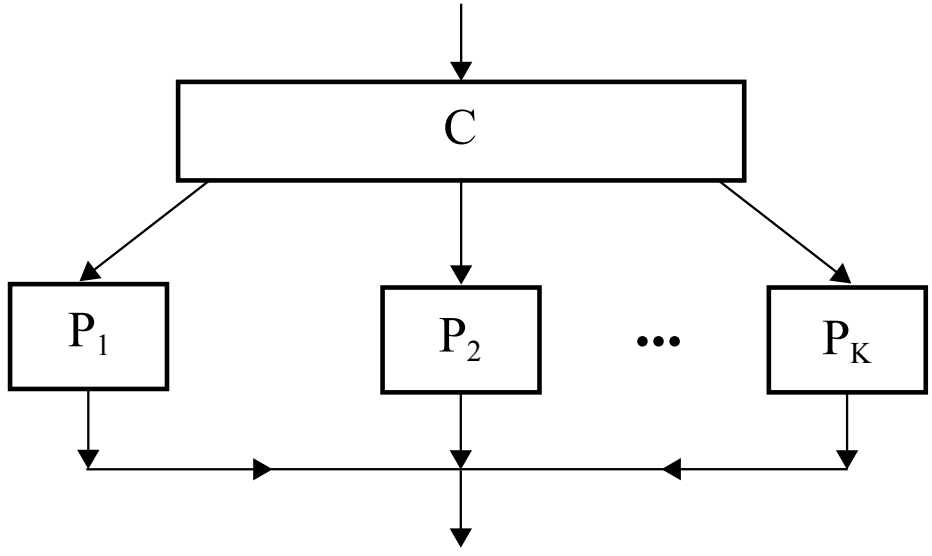
IF C THEN P



IF C THEN P_1 ELSE P_2

Basis of Structured Programming

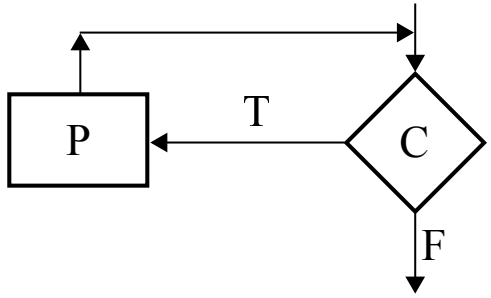
Selection:



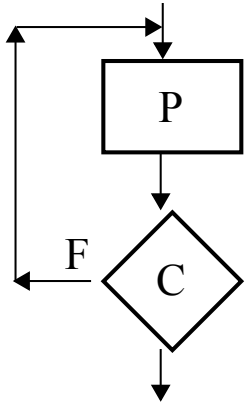
CASE C OF P_1, ... P_K

Basis of Structured Programming

Iteration



WHILE C DO P



REPEAT P UNTIL C

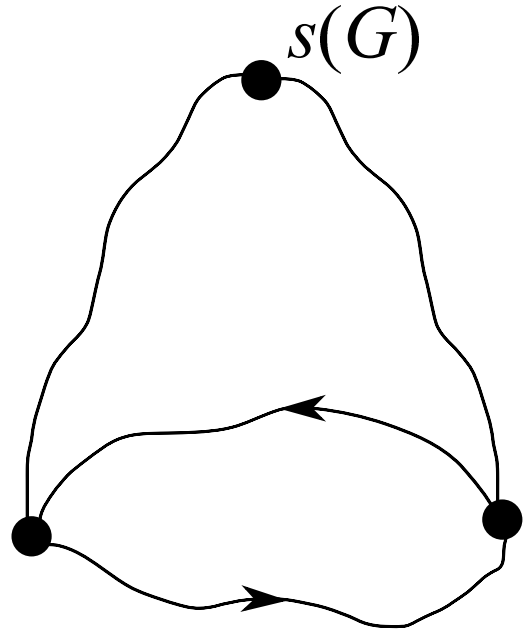
Our Problem

Given a computer program (flowcharts) is it structured ?

Approach: graph theory

Flow graphs and reducible graphs

- A **flow graph** is a directed graph G containing one source $s(G)$
- A **reducible graph** is a flow graph G such that every cycle C contains a unique entry point, for paths starting in $s(G)$

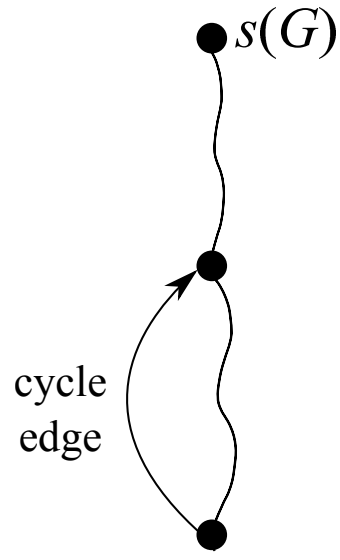


Flow graphs and reducible graphs

- Every algorithm (structured or not) corresponds to a flow graph.
- Every structured algorithm corresponds to a reducible graphs.

Reducible graphs

- Characterizations: Hecht and Ullman (1970, 1974), Tarjan (1974)
- Efficient Recognition: Tarjan (1974)
- Useful characterization: A graph G is reducible if and only if any DFS of G, starting from $s(G)$ determines the same set of cycle edges.

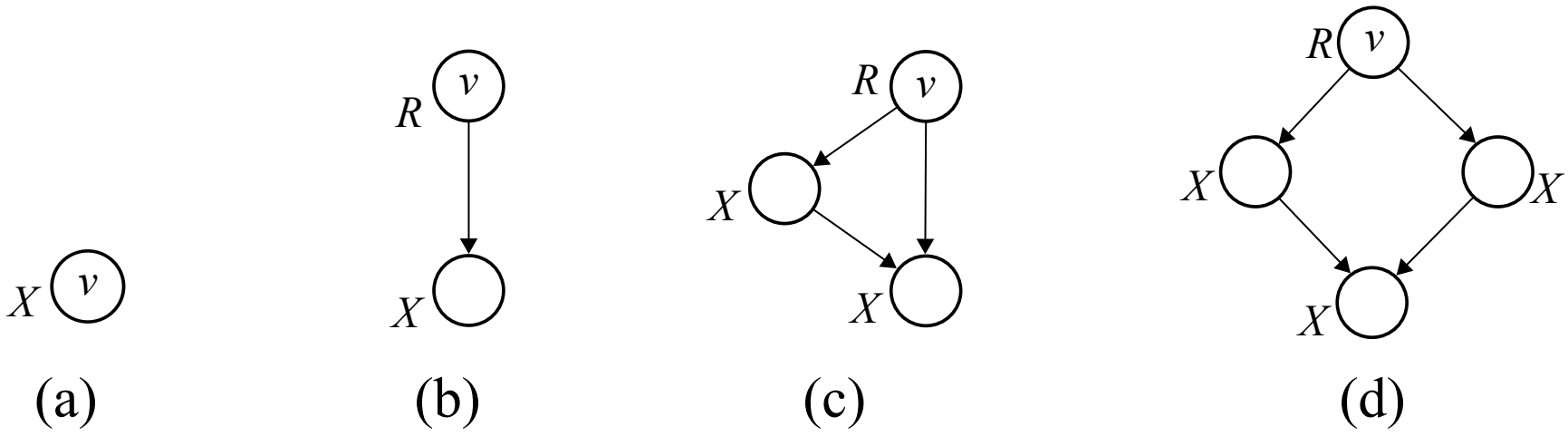


Statement graphs

A **statement graph** is a directed graph H, satisfying

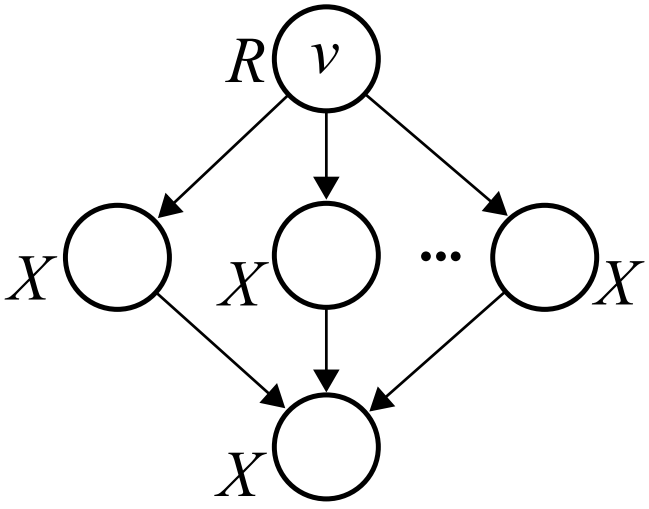
- Each vertex is labelled as **regular** (R) or **expansive** (X)
- H belongs to one of the following classes:
  - *trivial* graph
  - *sequence* graph
  - *if* graph
  - *if-then-else* graph
  - p*-case* graph, $p \geq 3$
  - *while* graph
  - *repeat* graph

Statement graphs

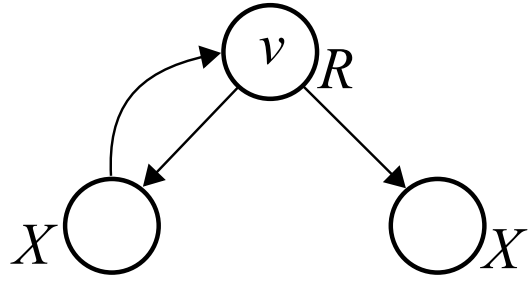


Note: All single source-sink

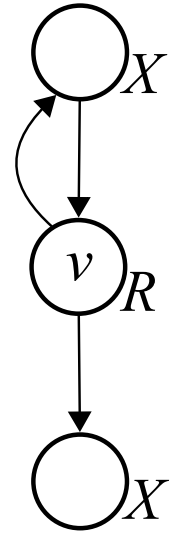
Statement graphs



(e)



(f)



(g)

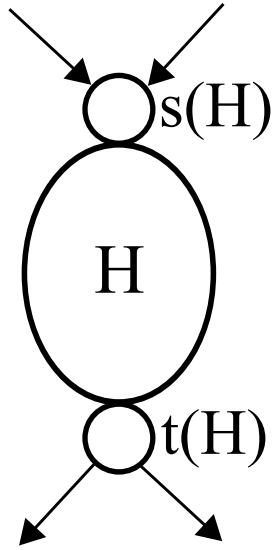
Note: All single source-sink

Closed subgraphs

Let H, single source-sink subgraph of G

H **closed** when

1. $v \in V(H) \setminus s(H) \Rightarrow N^-[v] \subseteq V(H)$
2. $v \in V(H) \setminus t(H) \Rightarrow N^+[v] \subseteq V(H)$
3. $vs(H)$ is a cycle edge $\Rightarrow v \in N^+(s(H))$



Prime subgraphs

Let H be an induced subgraph of G
H is **prime** when H is

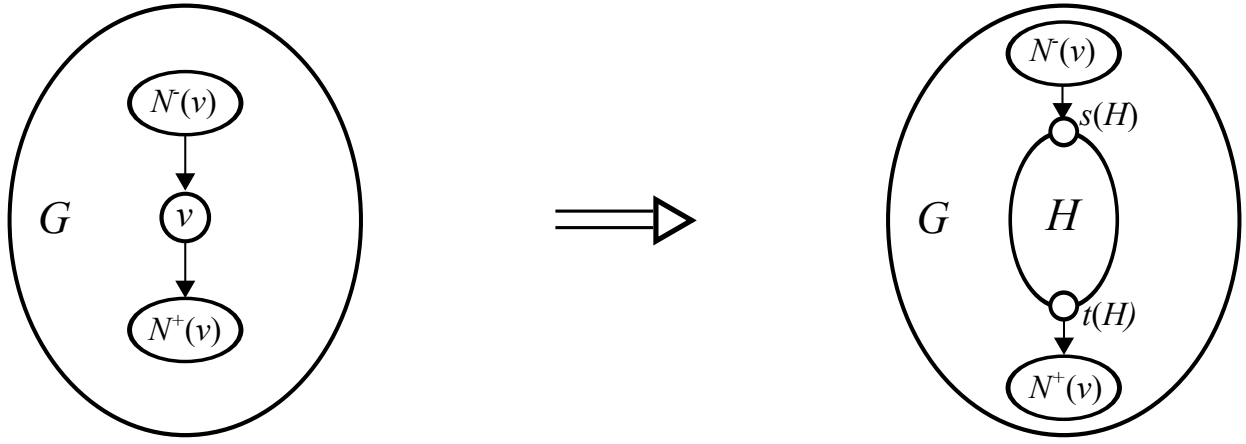
1. isomorphic to some non-trivial statement graph
2. closed

NOTE: The **while** and **repeat** graphs might be isomorphic as statement graphs, but distinguishable as prime subgraphs, even with no labels.

Expansion operation

Let G, H graphs, $V(G) \cap V(H) = \emptyset$, H single source-sink, $v \in V(G)$.
The **expansion of** v **into** H consists of replacing v by H, in G, such that

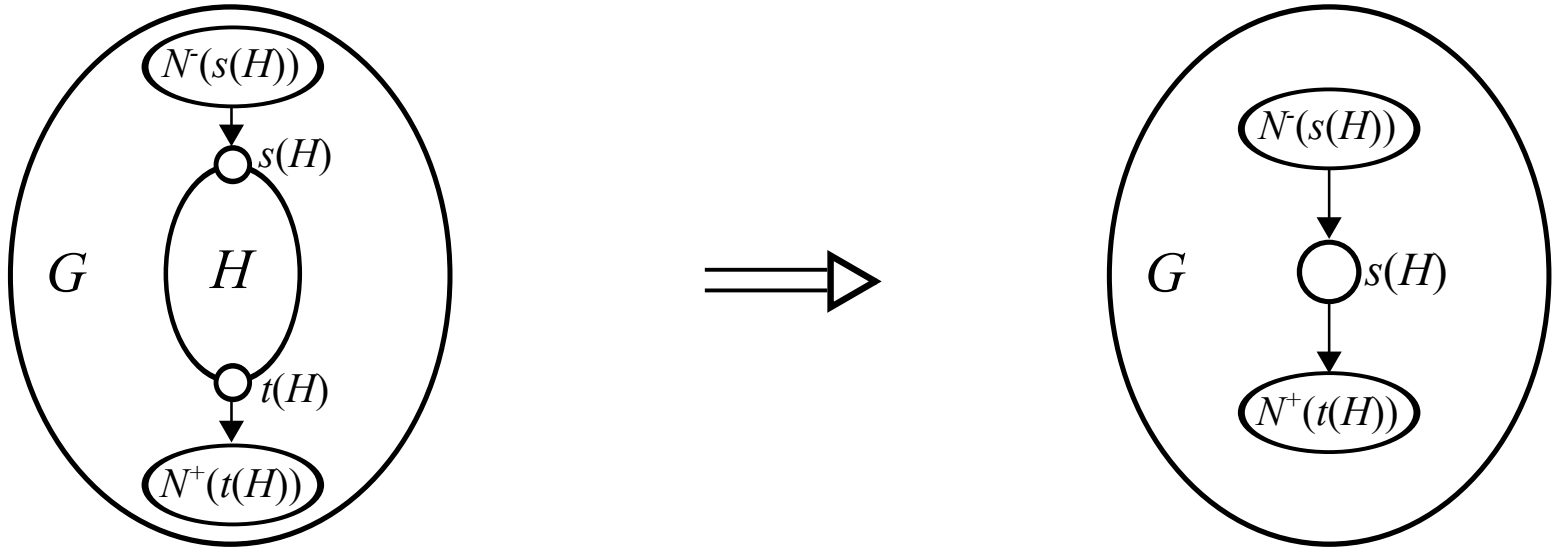
- $N_G^-(s(H)) := N_G^-(v)$
- $N_G^+(t(H)) := N_G^+(v)$
- Remaining adjacencies unchanged



Contraction operation

G graph, H prime subgraph of G
The contraction of H into a single vertex:

- Identifies the vertices if H into the source $s(H)$ of H
- Remove possible parallel edges or loops.

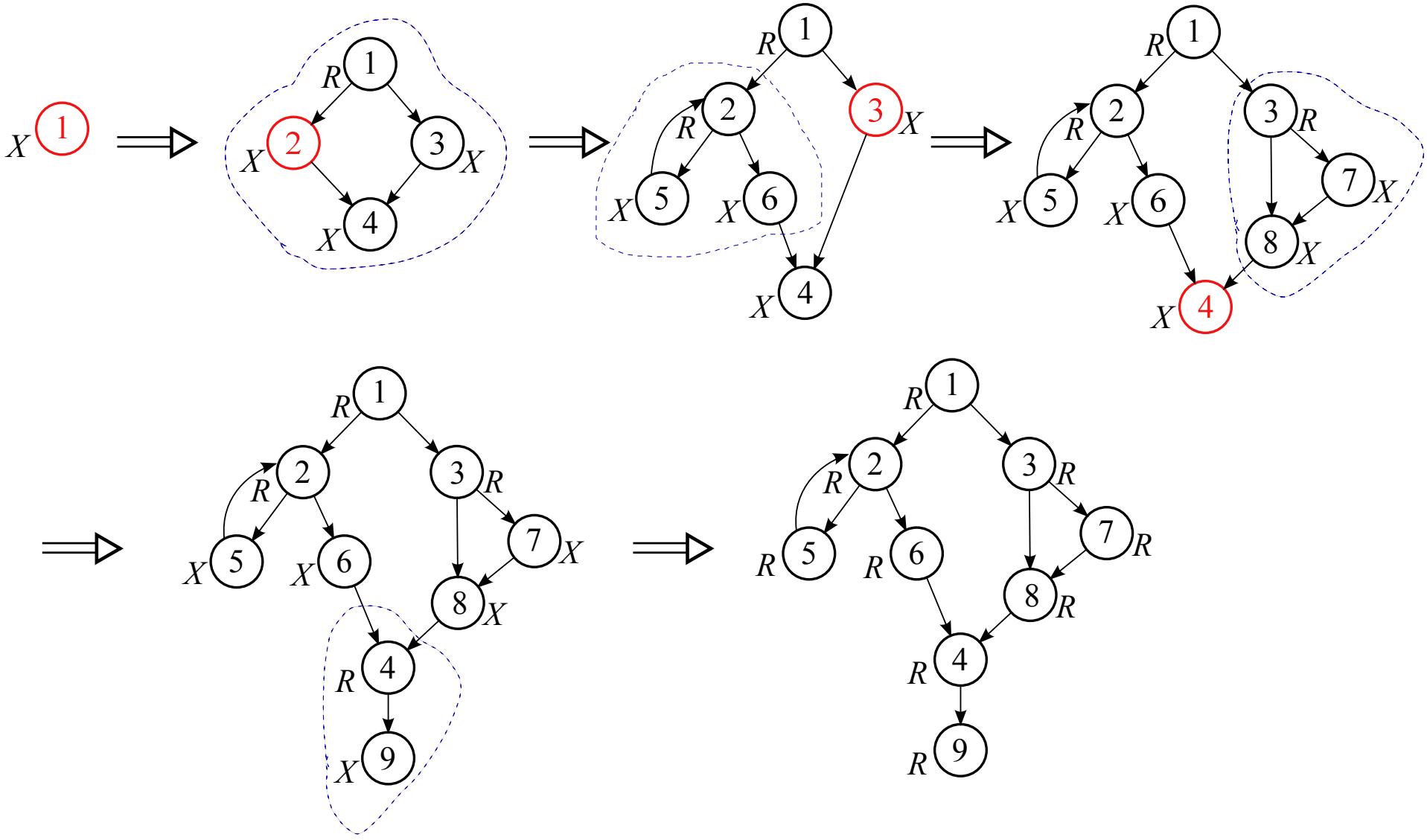


Dijkstra graph

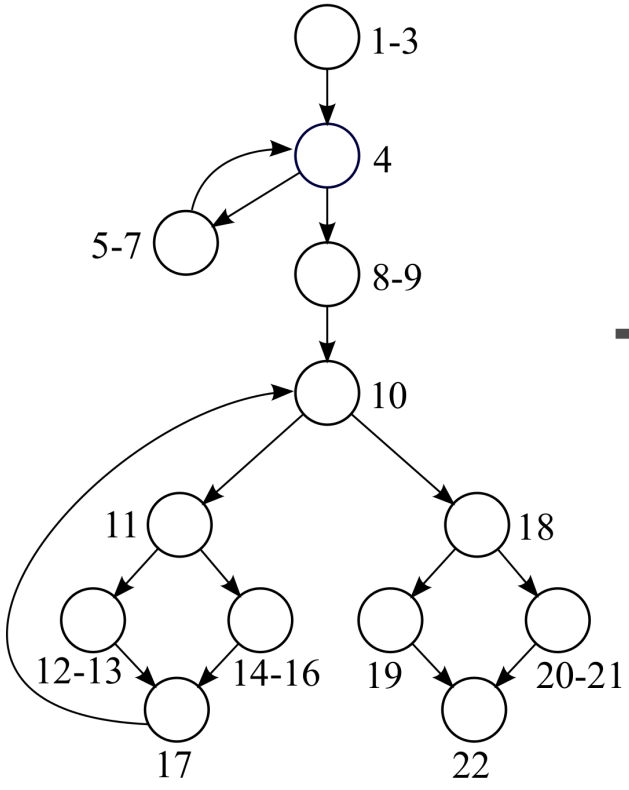
A **A Dijkstra graph (DG)** is a graph with vertices labelled as X or R, recursively defined as:

1. A trivial statement graph is a DG
2. Any graph obtained from a DG, by expanding some X-vertex of it into a statement graph is also a DG. Furthermore, after expanding an X-labelled vertex into H, the vertex $s(H)$ is labelled as R.

Example


X 1
R 1
X 2
3 X
X 4
R 1
R 2
3 X
X 5
X 6
X 4
R 1
R 2
3 R
X 5
X 6
7 X
8 X
X 4
R 1
R 2
3 R
X 5
X 6
7 X
8 X
R 4
X 9
R 1
R 2
3 R
R 5
R 6
7 R
8 R
R 4
R 9

Example: Binary search




```
   void main()
   {
1     int c, first, last, middle, n, search, array[100];
2     scanf("%d",&n); //number of elements
3     c = 0;
4     while (c < n){
5        scanf("%d",&array[c]);
6        c++;
7     }
8     scanf("%d",&search); //value to find
9     first = 0; last = n - 1; middle = (first+last)/2;
10    while( array[middle] != search && first <= last ){
11       if ( array[middle] < search ){
12          first = middle + 1;
13          middle = (first + last)/2;}
14       else{
15          last = middle - 1;
16          middle = (first + last)/2;}
17    }
18    if (item == a[mid]) {
19       printf("Binary search successfull!!\n")}
20    else {
21       printf("Search failed! \n")}
22 }
```

Constructing Dijkstra graphs

Theorem 1 *A graph G is a DG if and only if there is a sequence of graphs $G_0, \ldots, G_k$ each of them having the vertices labelled as X or R, such that*

1. *G_0 is the trivial graph, with the vertex labelled X*
2. *$G_k \cong G$*
3. *G_i is obtained from G_{i-1}, $i \geq 1$, by expanding some X-vertex of it into a non-trivial statement graph, whose source $s(H)$ receives label R.*

Still: How to recognize ?

Basic properties

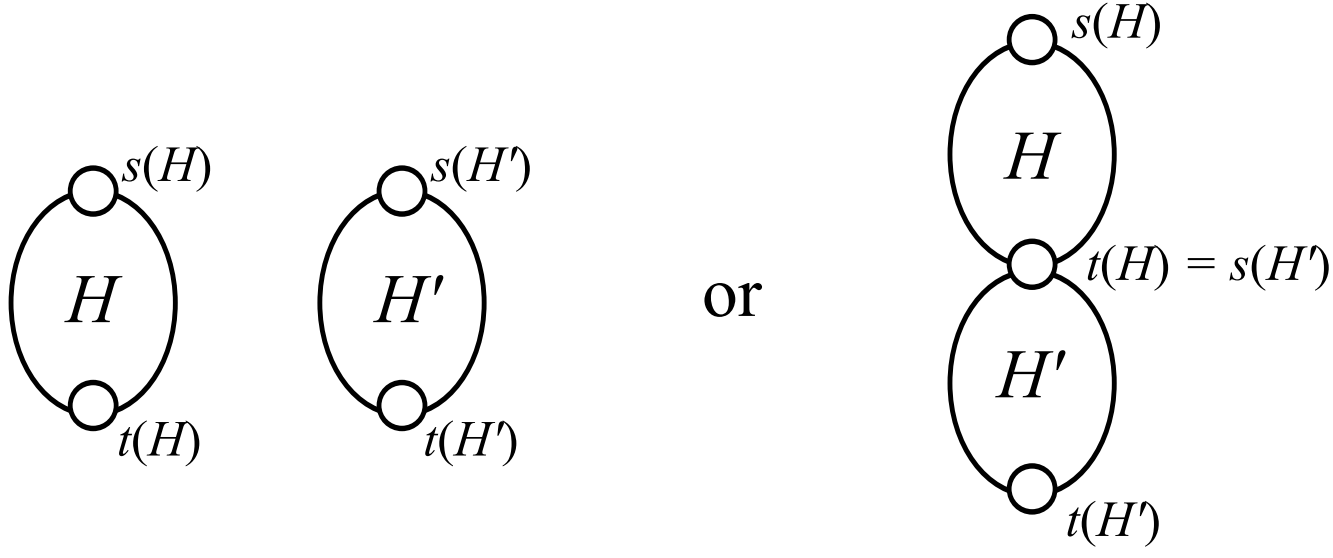
Lemma 1 *Let G be a DG. Then*

1. *G contains some prime subgraph*
2. *G is single a source-sink graph*
3. *G is a reducible flow graph*

Independence

$\mathcal{H}(G) =$ set of non-trivial prime subgraphs of G,
Let $H, H' \in \mathcal{H}$. H, H' are **independent** when:

1. $V(H) \cap V(H') = \emptyset$, or
2. $V(H) \cap V(H') = \{v\}$, where
   $v = s(H) = t(H')$, or $v = s(H') = t(H)$



Independent primes

Lemma 2 *Let H, H' be two distinct prime subgraphs of some graph G. Then H, H' are independent.*

Images

G graph,
$\mathcal{H}(G) =$ set of non-trivial prime subgraphs of G,
$H \in \mathcal{H}(G)$
$G \downarrow H =$ graph obtained from G by contracting H

Image of a vertex:
For $v \in V(G)$, the **image** of v in $G \downarrow H =$

$$I_{G\downarrow H}(v) = \begin{cases} v & \text{, if } v \notin V(H) \\ s(H) & \text{, otherwise.} \end{cases}$$

Images

Image of a subset of vertices:
For $V' \subseteq V(G)$, the **(subset) image** of V' in $G \downarrow H =$

$I_{G\downarrow H}(V') = \cup_{v \in V'} I_{G\downarrow H}(v)$

Image of subgraph:
For $H' \subseteq G$, the **(subgraph) image** of H' in $G \downarrow H =$

$I_{G\downarrow H}(H') =$ subgraph induced in $G \downarrow H$ by the subset of vertices $I_{G\downarrow H}(V(H'))$

Preservation of Primes

G, arbitrary graph
$H, H' \in \mathcal{H}(G)$, $H \neq H'$.

Lemma 3 $I_{G\downarrow H}(H') \in \mathcal{H}(G \downarrow H)$

Commutative law

Lemma 4 $H, H' \in \mathcal{H}(G)$

$$(G \downarrow H) \downarrow (I_{G\downarrow H}(H')) \cong (G \downarrow H') \downarrow (I_{G\downarrow H'}(H))$$

Contractile sequences

A sequence of graphs $G_0, \ldots, G_k$ is a **contractile sequence** for a graph G, when

1. $G \cong G_0$,
2. $G_{i+1} \cong G_i \downarrow H_i$, for some $H_i \in \mathcal{H}(G_i)$, $i < k$. Call H_i, the **contracting prime** of G_i.

$G_0, \ldots, G_k$ is **maximal** when $\mathcal{H}(G_k) = \emptyset$.

In particular, if G_k is the trivial graph then $G_0, \ldots, G_k$ is maximal.

Iterated images

$G_0, \ldots, G_k$, contractile sequence of G
H_j, contracting prime of G_j
(i.e. $G_{j+1} \cong G_j \downarrow H_j$)

For $H'_j \subseteq G_j$, the **iterated image** of H'_j in G_q, $q \geq j$ is

$$I_{G_q}(H'_j) = \begin{cases} H'_j, & \text{if } q = j \\ I_{G_q}(I_{G_{j+1}}(H'_j)), & \text{otherwise.} \end{cases}$$

Main characterization

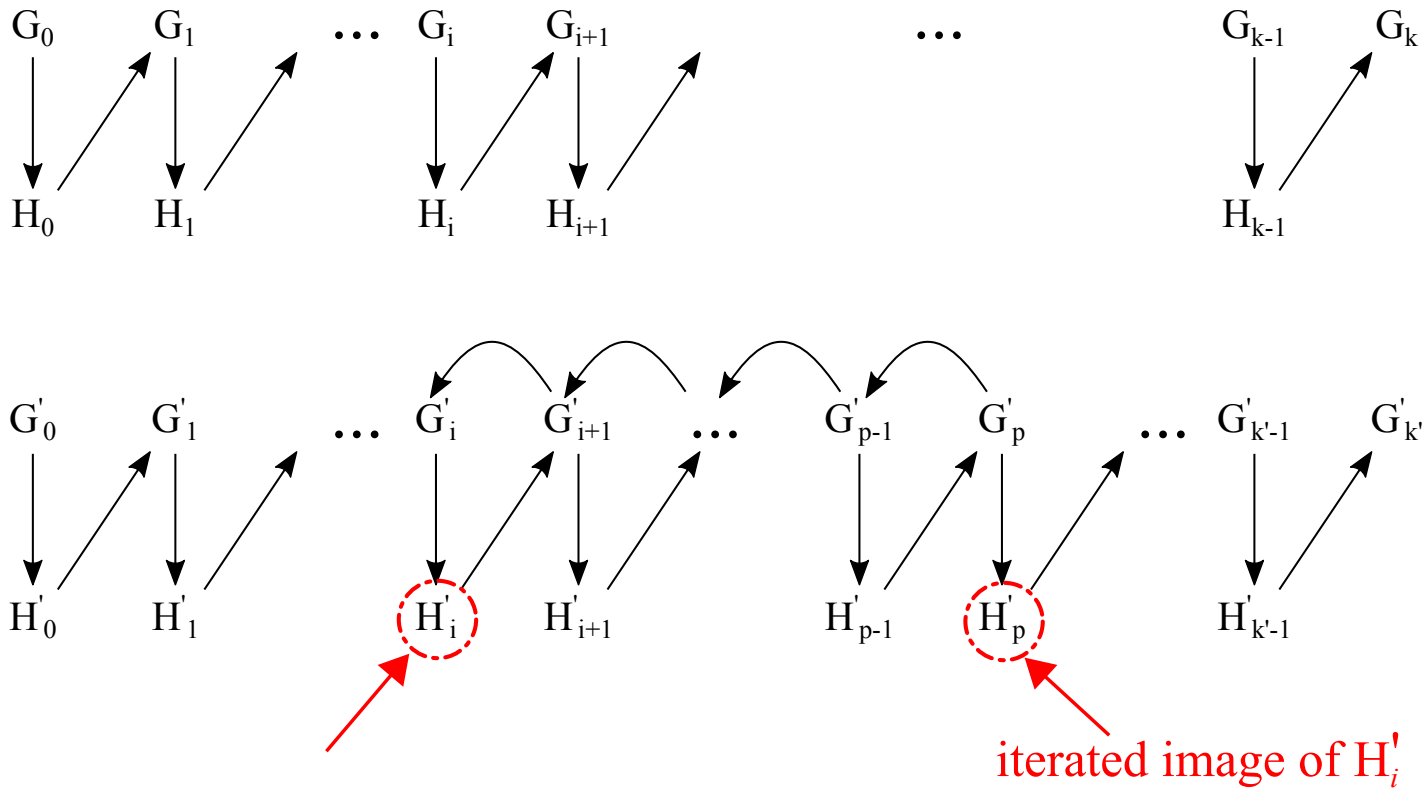
Theorem 2 *Let G be an arbitrary graph, with $G_0, \ldots, G_k$ and $G'_0, \ldots, G'_{k'}$ two contractile sequences of G. Then G_k and $G'_{k'}$ are isomorphic. Furthermore, $k = k'$.*

Notes:

1) Leads to a greedy recognition algorithm.

2) The input graph G has no labels.

sketch



$G_{j+1} \cong G_h \downarrow H_j$ and $G'_{j+1} \cong G_h \downarrow H_j$

$G_0 \cong G'_0, G_1 \cong G'_1, \ldots, G_i \cong G'_i, G_{i+1} \ncong G'_{i+1} \Rightarrow$
$H_0 \cong H'_0, H_1 \cong H'_1, \ldots, H_i \ncong H'_i$

Recognition

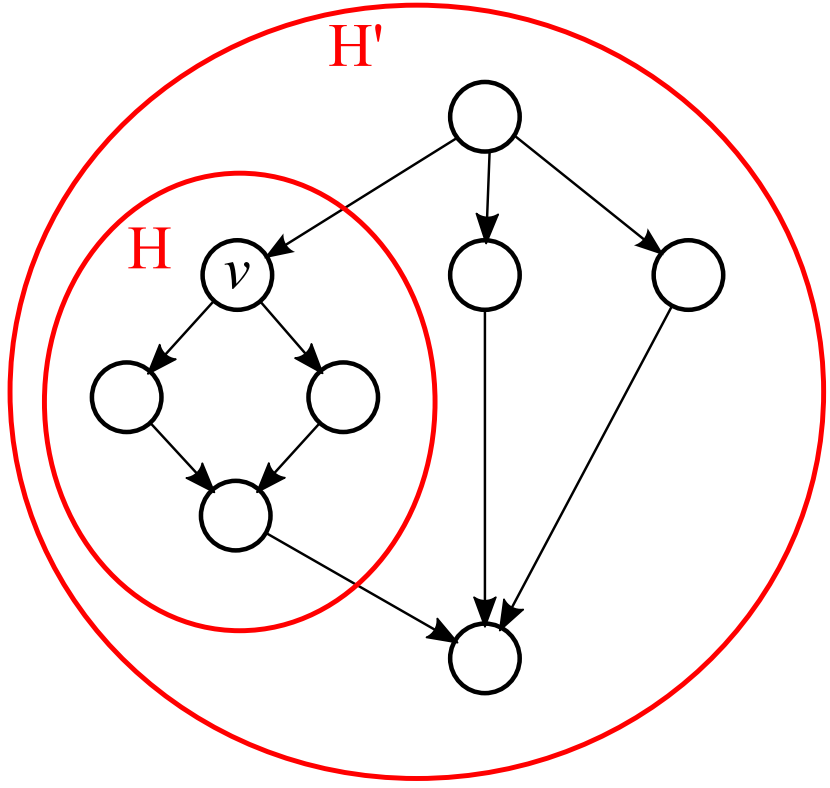
Corollary 1 *Let G be an arbitrary graph, and $G_0, \ldots, G_k$ a contractile sequence of it. Then G is a DG if and only if G_k is a trivial graph.*

Bottom-up contractile sequences

- G, reducible graph
- $G_0, \ldots, G_k$, contractile sequence $\mathcal{C}$ of G
- H_i the contracting prime of G_i, $0 \leq i < k$
- $\mathcal{C}$ is a **bottom-up (contractile) sequence** of G when each contracting prime H_i satisfies:
  $s(H_i)$ is not a descendant of $s(H)$, for any prime $H \neq H_i$ of G_i.

Generating primes by contractions

Contracting a prime H of G may generate a (new) prime H' of $G \downarrow H$.



Generator and generated

Relation between generator and generated primes:

Lemma 5 *G, reducible graph*
$H \in \mathcal{H}(G)$, $H' \in \mathcal{H}(G \downarrow H) \setminus \mathcal{H}(G)$, $v = s(H)$.
Then v is a descendant of $s(H')$ in $G \downarrow H$.

Recognition Algorithm

- G, reducible graph
- Iteratively, find a lowest vertex v of G, s.t.
- v is the source of a prime subgraph H of G
- then contract H.
- Stop when no primes exist any more.

The initial graph is DG iff the final is trivial

The algorithm

G, arbitrary flow graph (no labels)
E_C, set of cycle edges of a DFS of G, starting at $s(G)$
$v_1, \ldots, v_n$, topological sorting of $G - E_C$
$i := n$
while $i \geq 1$ **do**
    if G is the trivial graph
        then return YES, **stop**
    if v_i is the source of a prime subgraph H of G
        then $G := G \downarrow H$
    $i := i - 1$

return NO, **stop**

Avoiding to recognize reducible graphs

Lemma 6 *Let G be an arbitrary flow graph. If the algorithm recognizes G as a Dijkstra graph then necessarily G is a reducible graph.*

Complexity

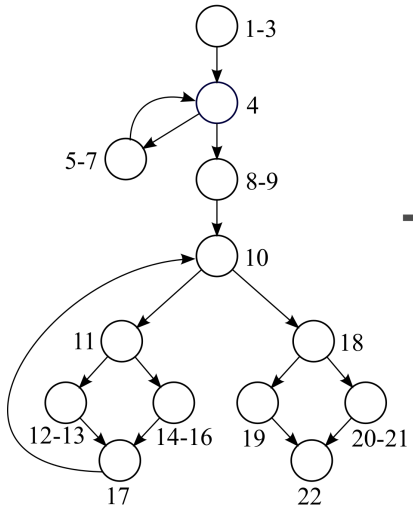
- Topological sorting: $O(m)$
- Deciding if vertex v_i is the source of a prime: $O(|N^+(v_i)|)$
- There can be $O(n)$ primes
- Contracting a prime: $O(|N^+(v_i)|)$
- Each edge is traversed only a constant number of times, overall
- Complexity: $O(m)$

Bounding the size of a DG

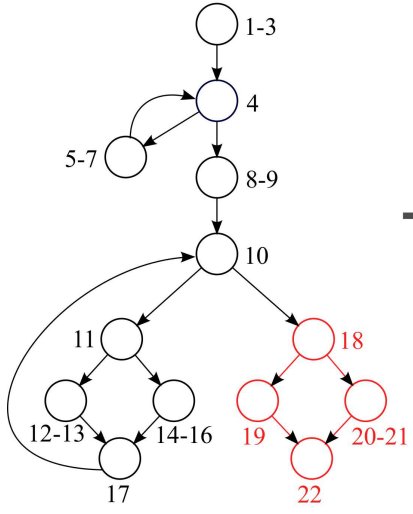
Lemma 7 *Let G be a DG graph.*
Then $m \leq 2n - 2$.

Consequence:
Complexity of the recognition algorithm:

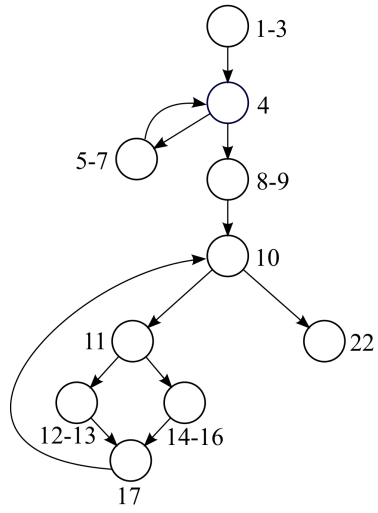
$O(n)$



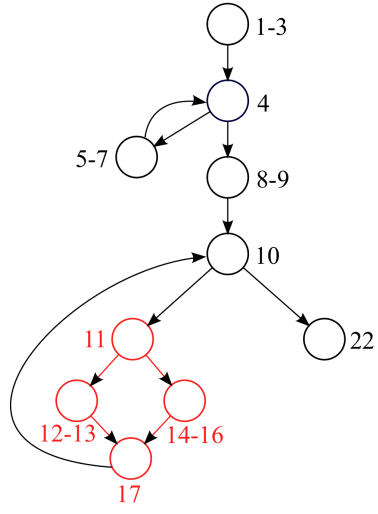

```
  void main()
  {
1    int c, first, last, middle, n, search, array[100];
2    scanf("%d",&n); //number of elements
3    c = 0;
4    while (c < n){
5       scanf("%d",&array[c]);
6       c++;
7    }
8    scanf("%d",&search); //value to find
9    first = 0; last = n - 1; middle = (first+last)/2;
10   while( array[middle] != search && first <= last ){
11      if ( array[middle] < search ){
12         first = middle + 1;
13         middle = (first + last)/2;}
14      else{
15         last = middle - 1;
16         middle = (first + last)/2;}
17   }
18   if (item == a[mid]) {
19      printf("Binary search successfull!!\n")}
20   else {
21      printf("Search failed! \n")}
22 }
```



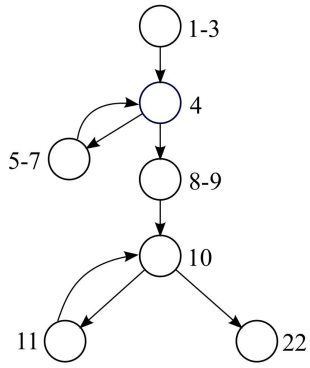

```
  void main()
  {
1    int c, first, last, middle, n, search, array[100];
2    scanf("%d",&n); //number of elements
3    c = 0;
4    while (c < n){
5       scanf("%d",&array[c]);
6       c++;
7    }
8    scanf("%d",&search); //value to find
9    first = 0; last = n - 1; middle = (first+last)/2;
10   while( array[middle] != search && first <= last ){
11      if ( array[middle] < search ){
12         first = middle + 1;
13         middle = (first + last)/2;}
14      else{
15         last = middle - 1;
16         middle = (first + last)/2;}
17   }
18   if (item == a[mid]) {
19      printf("Binary search successfull!!\n")}
20   else {
21      printf("Search failed! \n")}
22 }
```



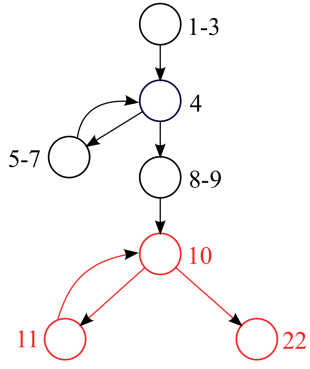

```
  void main()
  {
1    int c, first, last, middle, n, search, array[100];
2    scanf("%d",&n); //number of elements
3    c = 0;
4    while (c < n){
5       scanf("%d",&array[c]);
6       c++;
7    }
8    scanf("%d",&search); //value to find
9    first = 0; last = n - 1; middle = (first+last)/2;
10   while( array[middle] != search && first <= last ){
11      if ( array[middle] < search ){
12         first = middle + 1;
13         middle = (first + last)/2;}
14      else{
15         last = middle - 1;
16         middle = (first + last)/2;}
17   }
18
19
20
21
22 }
```



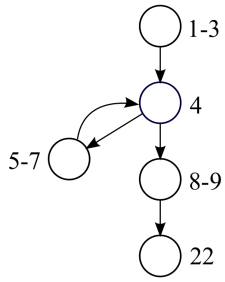

```
  void main()
  {
1    int c, first, last, middle, n, search, array[100];
2    scanf("%d",&n); //number of elements
3    c = 0;
4    while (c < n){
5       scanf("%d",&array[c]);
6       c++;
7    }
8    scanf("%d",&search); //value to find
9    first = 0; last = n - 1; middle = (first+last)/2;
10   while( array[middle] != search && first <= last ){
11      if ( array[middle] < search ){
12         first = middle + 1;
13         middle = (first + last)/2;}
14      else{
15         last = middle - 1;
16         middle = (first + last)/2;}
17   }
18
19
20
21
22 }
```



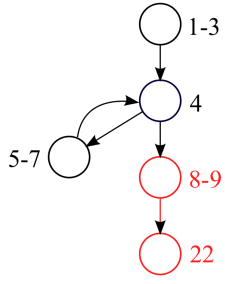

```
  void main()
  {
1    int c, first, last, middle, n, search, array[100];
2    scanf("%d",&n); //number of elements
3    c = 0;
4    while (c < n){
5       scanf("%d",&array[c]);
6       c++;
7    }
8    scanf("%d",&search); //value to find
9    first = 0; last = n - 1; middle = (first+last)/2;
10   while( array[middle] != search && first <= last ){
11
12
13
14
15
16
17   }
18
19
20
21
22 }
```



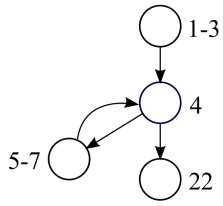

```
  void main()
  {
1    int c, first, last, middle, n, search, array[100];
2    scanf("%d",&n); //number of elements
3    c = 0;
4    while (c < n){
5       scanf("%d",&array[c]);
6       c++;
7    }
8    scanf("%d",&search); //value to find
9    first = 0; last = n - 1; middle = (first+last)/2;
10   while( array[middle] != search && first <= last ){
11
12
13
14
15
16
17   }
18
19
20
21
22 }
```



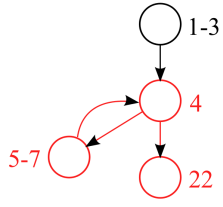

```
  void main()
  {
1    int c, first, last, middle, n, search, array[100];
2    scanf("%d",&n); //number of elements
3    c = 0;
4    while (c < n){
5       scanf("%d",&array[c]);
6       c++;
7    }
8    scanf("%d",&search); //value to find
9    first = 0; last = n - 1; middle = (first+last)/2;
10
11
12
13
14
15
16
17
18
19
20
21
22 }
```



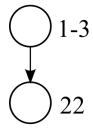

```
  void main()
  {
1    int c, first, last, middle, n, search, array[100];
2    scanf("%d",&n); //number of elements
3    c = 0;
4    while (c < n){
5       scanf("%d",&array[c]);
6       c++;
7    }
8    scanf("%d",&search); //value to find
9    first = 0; last = n - 1; middle = (first+last)/2;
10
11
12
13
14
15
16
17
18
19
20
21
22 }
```



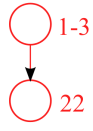

```
  void main()
  {
1    int c, first, last, middle, n, search, array[100];
2    scanf("%d",&n); //number of elements
3    c = 0;
4    while (c < n){
5       scanf("%d",&array[c]);
6       c++;
7    }
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22 }
```




```
  void main()
  {
1    int c, first, last, middle, n, search, array[100];
2    scanf("%d",&n); //number of elements
3    c = 0;
4    while (c < n){
5       scanf("%d",&array[c]);
6       c++;
7    }
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22 }
```



```
  void main()
  {
1    int c, first, last, middle, n, search, array[100];
2    scanf("%d",&n); //number of elements
3    c = 0;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22 }
```



```
  void main()
  {
1    int c, first, last, middle, n, search, array[100];
2    scanf("%d",&n); //number of elements
3    c = 0;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22 }
```

```
  void main()
  {
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22 }
```

Isomorphism of DG graphs

General idea:

- Define a code $C(G)$ for a DG graph G, such that $C(G)$ uniquely identifies G.
- $G_1 \cong G_2$ iff $C(G_1) = C(G_2)$
- $C(G)$ is a string of integers $\subseteq \{1, \ldots, \Delta(G) + 4\}$

Coding a DG graph

- Assign an integer, named $type(H)$, for each statement graph H,
- a code $C(v)$ for each vertex $v \in V(G)$,
- a code $C(H)$ for each prime subgraph H of a bottom-up contractile sequence of G.
- The code $C(G)$ of the graph is defined as $C(G) := C(s(G))$.
- For a subset $V' \subseteq V(G)$, the code $C(V')$ of V' is the set of strings $C(V') = \{C(v_i) | v_i \in V'\}$.
- Write $lex(C(V')) = C(v_1)||...||C(v_r)$, whenever $V' = \{v_1, \ldots, v_r\}$ and $C(v_i)$ is lexicographically not greater than $C(v_{i+1})$.

type(H); C(v), v=s(H); C(H)

statement graphs H	$type(H)$	$C(H)$
trivial	1	
sequence	2	$2 \| \| C(N^+(v))$
if-then	3	$3 \| \| C(N^+(v) \setminus N^{+2}(v)) \| \| C(N^{+2}(v))$
while	4	$4 \| \| C(N^+(v) \cap N^-(v)) \| \| C(N^+(v) \setminus N^-(v))$
repeat	5	$5 \| \| C(N^+(v) \| \| C(N^{+2}(v) \setminus \{v\})$
if-then-else	6	$6 \| \| lex(C(N^+(v))) \| \| C(N^{+2}(v))$
p-case	$p+4$	$p+4 \| \| lex(C(N^+(v))) \| \| C(N^{+2}(v))$

Setting the codes

- The types of statement graphs: table
- Encoding $C(v)$, $v \in V(G)$: initially set to 1.
- Subsequently, if v becomes the source of a prime graph H, assign $C(v) := C(v)||C(H)$
- Encoding $C(H)$: table
- $C(H)$ is written in terms of $type(H)$ and $C(w)|w \in V(H)$, and so iteratively.

Isomorphism algorithm

G, DG; E_C, set of cycle edges

Find a topological sorting $v_1, \ldots, v_n$ of $G - E_C$

for $i = n, n-1, \ldots, 1$ **do**

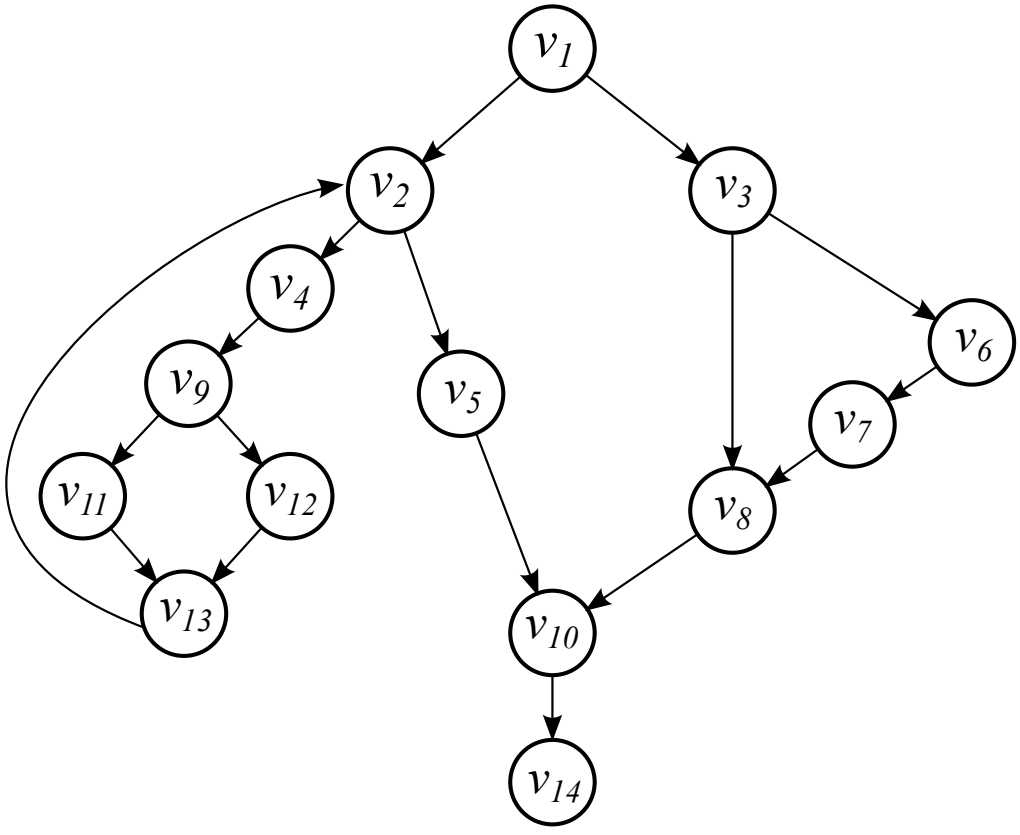
$C(v_i) := 1$

if v_i is the source of a prime subgraph H **then**

$$C(v_i) := C(v_i)|| \begin{cases} 2||C(N^+(v_i)), & \text{if } H \text{ is a sequence graph;} \\ 3||C(N^+(v_i) \setminus N^{+2}(v_i))||C(N^{+2}(v_i)), & \text{if } H \text{ is an if-then graph;} \\ 4||C(N^+(v_i) \cap N^-(v_i))||C(N^+(v_i) \setminus N^-(v_i)), & \\ & \text{if } H \text{ is a while graph,} \\ 5||C(N^+(v_i))||C(N^{+2}(v_i) \setminus \{v_i\}, & \\ & \text{if } H \text{ is a repeat graph;} \\ 6||lex(C(N^+(v_i)))||C(N^{+2}(v_i)), & \\ & \text{if } H \text{ is an if-then-else graph.} \\ p+4||lex(C(N^+(v_i)))||C(N^{+2}(v_i)), & \\ & \text{if } H \text{ is a p-case graph.} \end{cases}$$

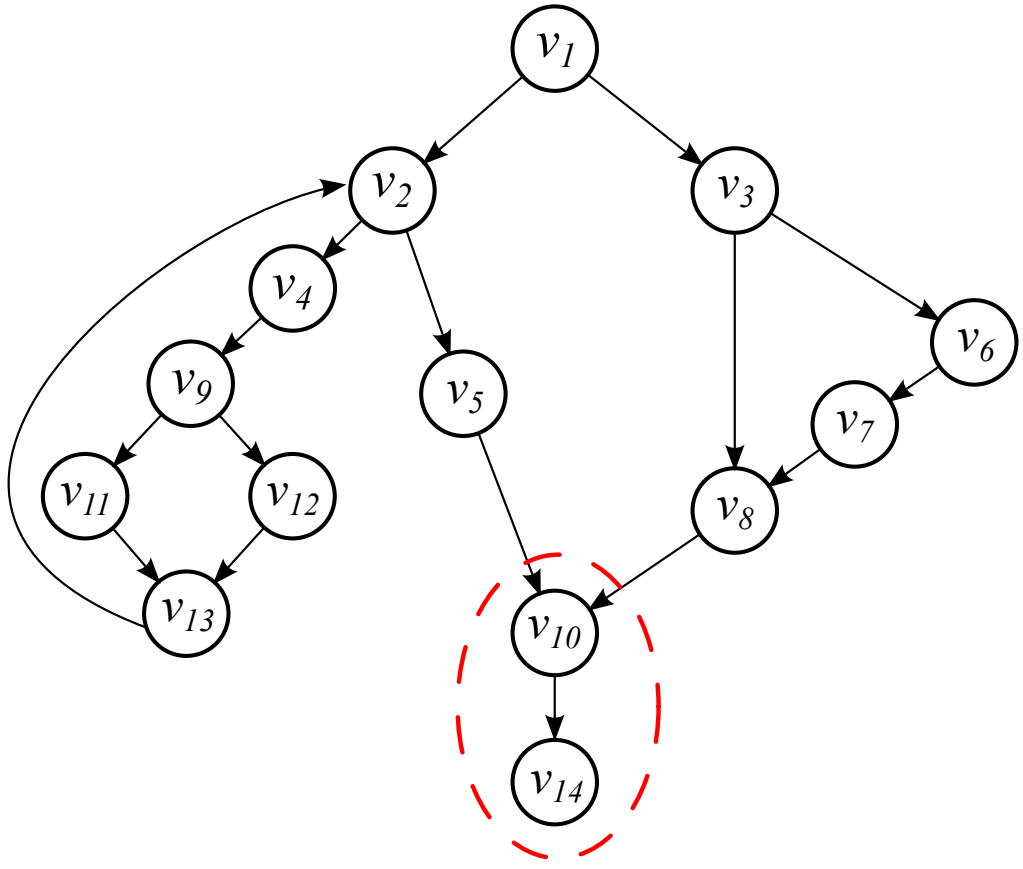
$C(G) := C(v_1)$

Example



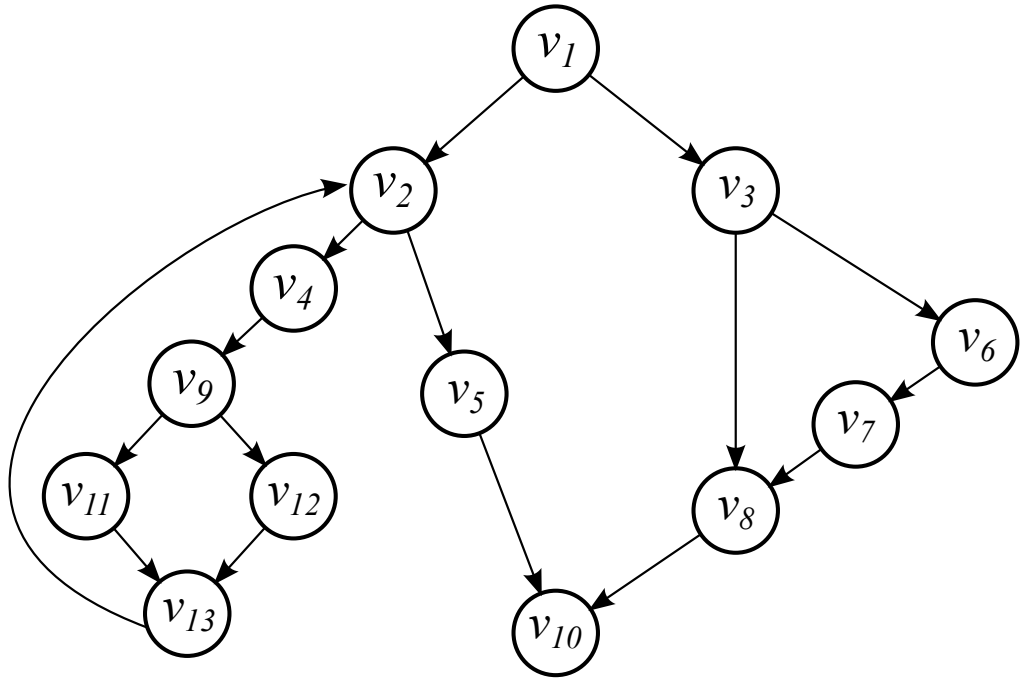
G, Dijkstra graph

Example

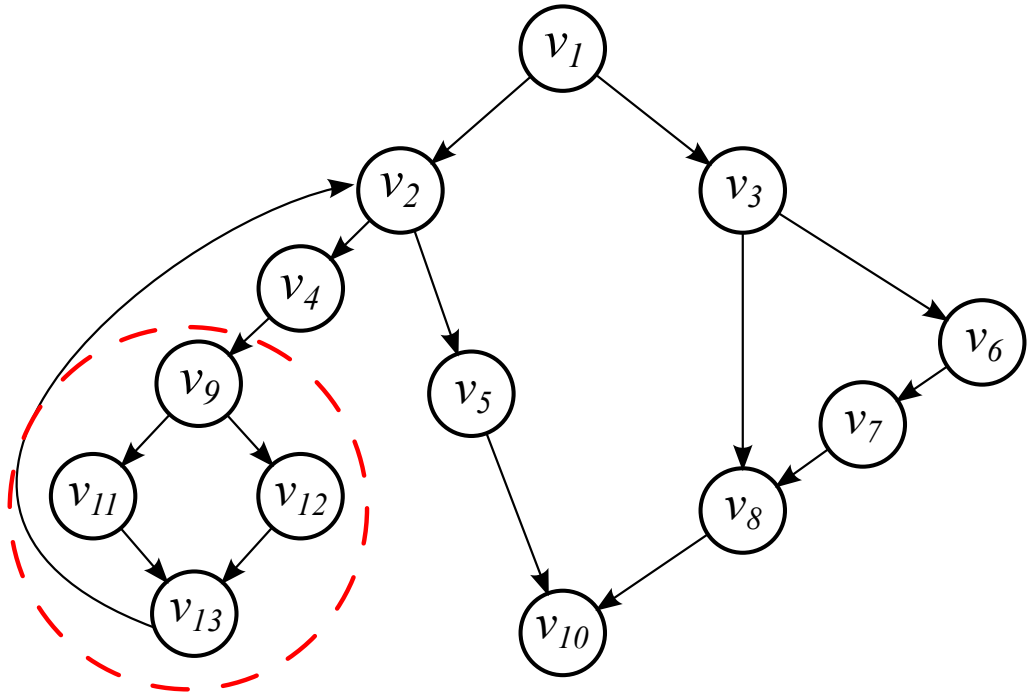


$$C(v_{10}) := 12||C(v_{14}) = 121$$

Example

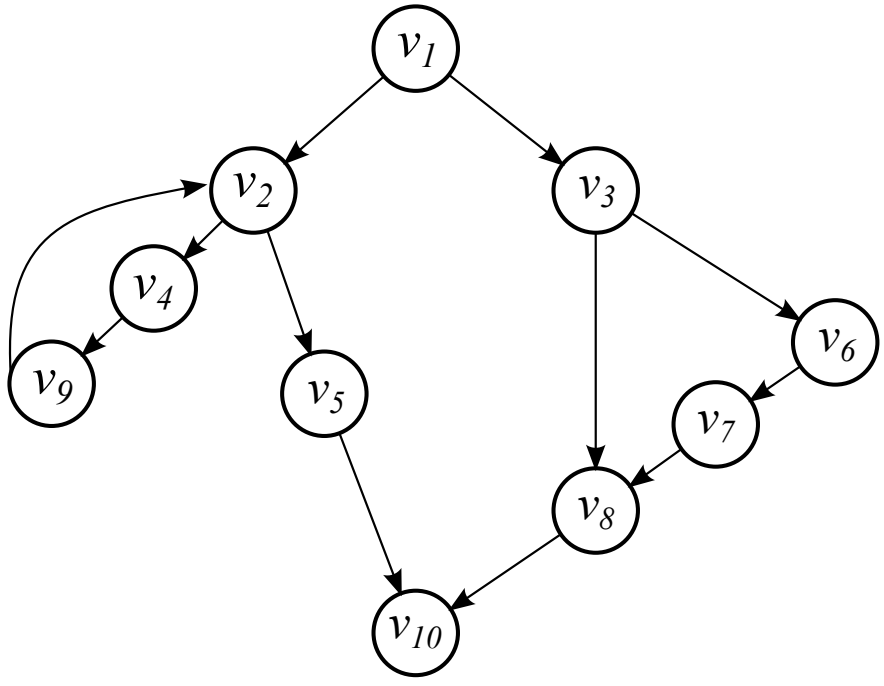


Example

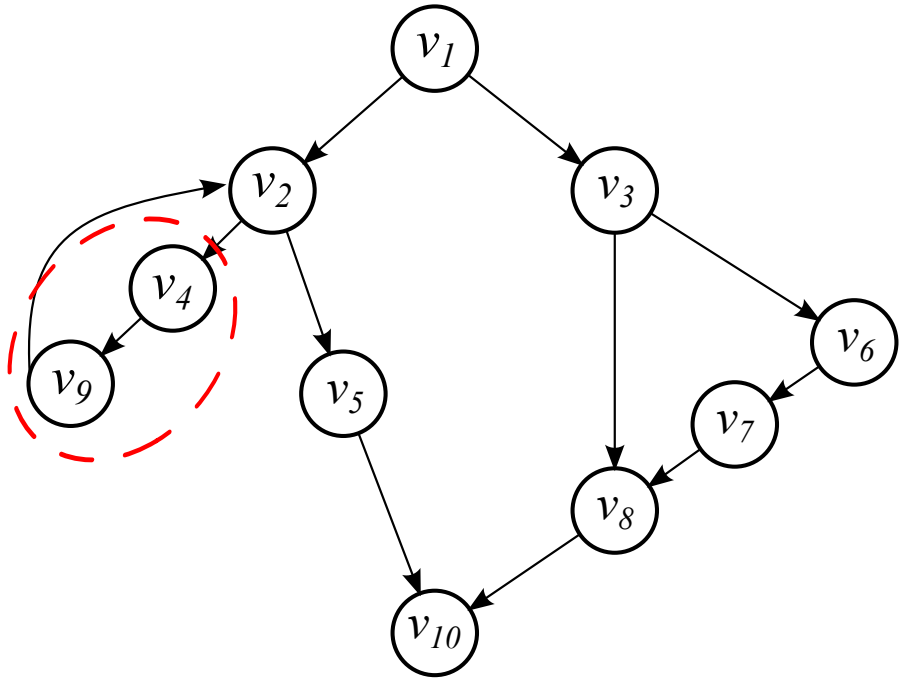


$$C(v_9) := 16||lex(C(v_{11}), C(v_{12}))||C(v_{13}) = 16111$$

Example

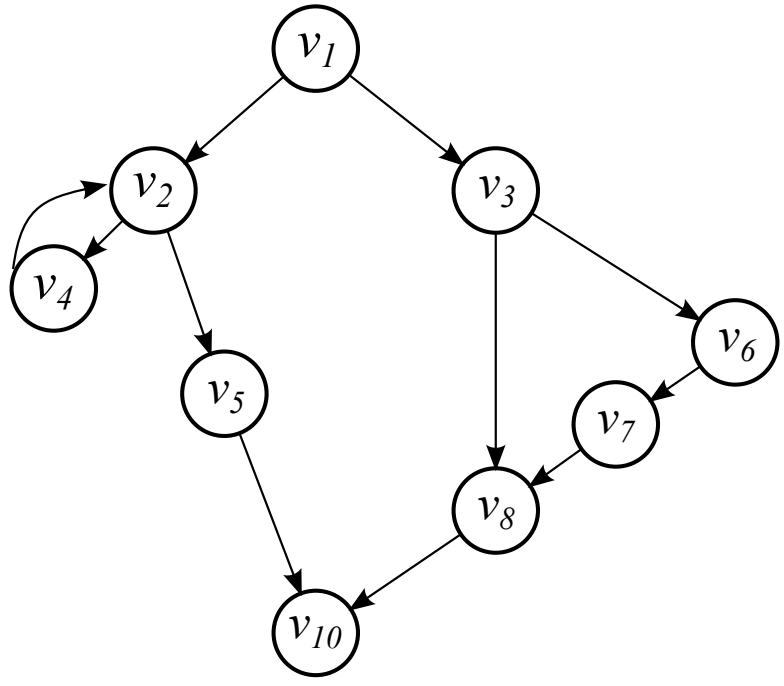


Example

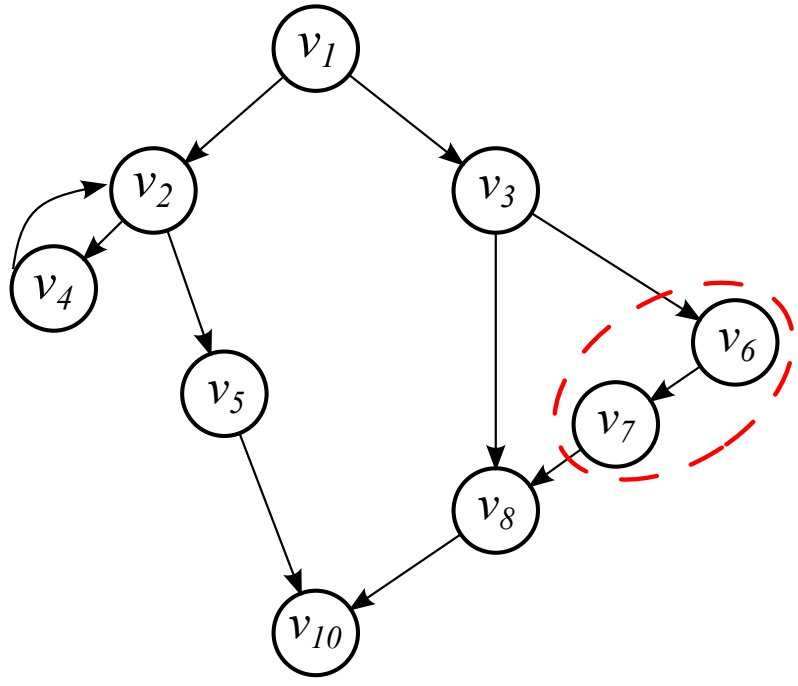


$$C(v_4) = 12||C(v_9) = 1216111$$

Example

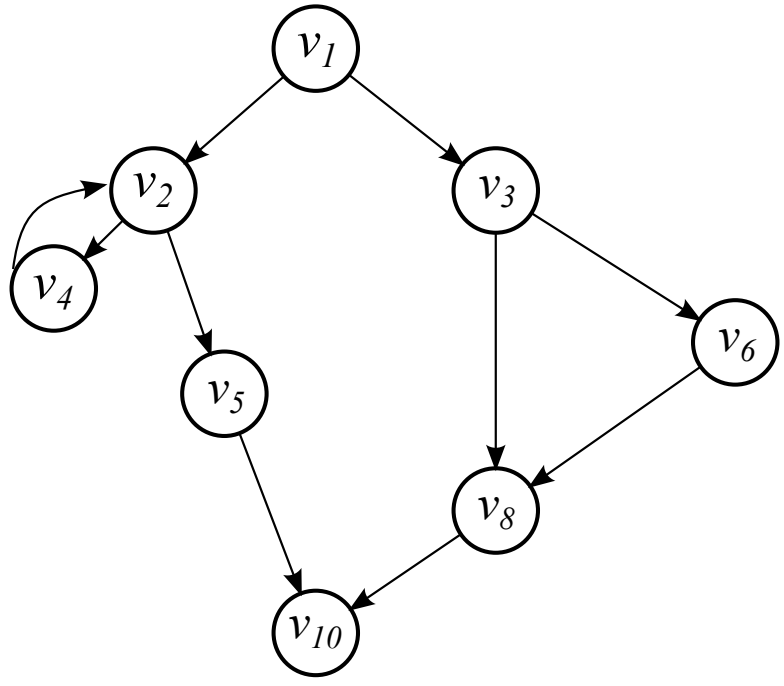


Example

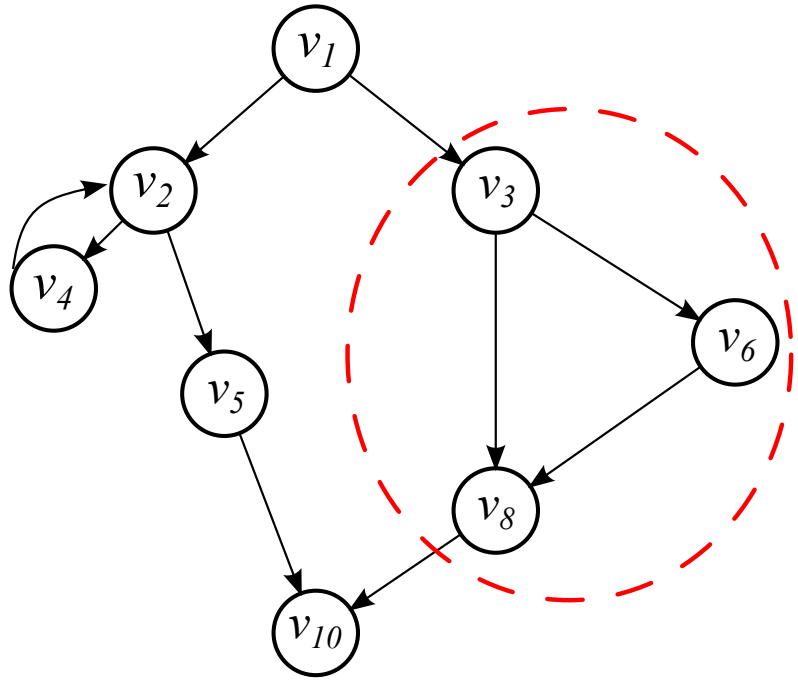


$$C(v_6) = 12||C(v_7) = 121$$

Example

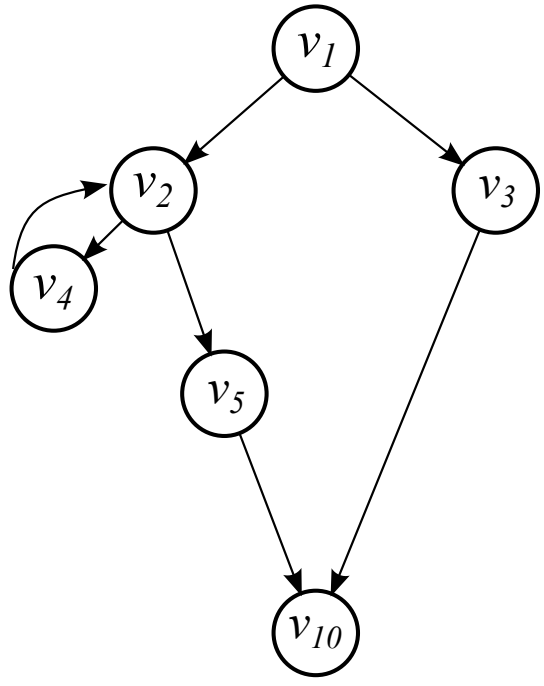


Example

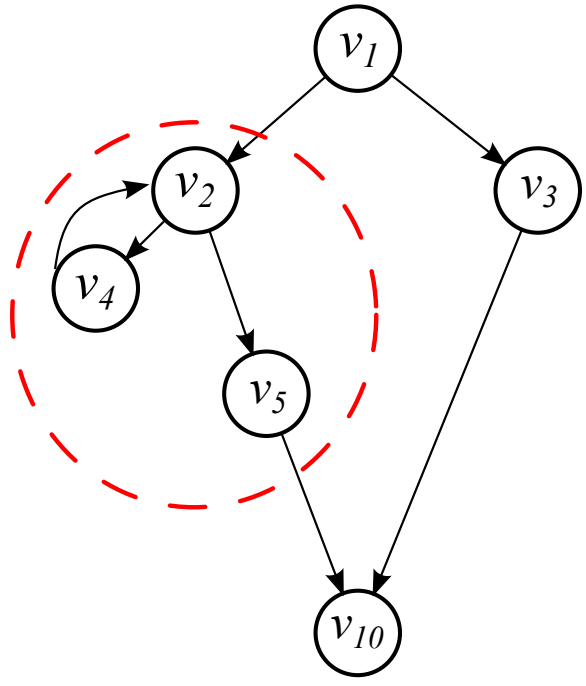


$$C(v_3) = 13||C(v_6)||C(v_8) = 131211$$

Example

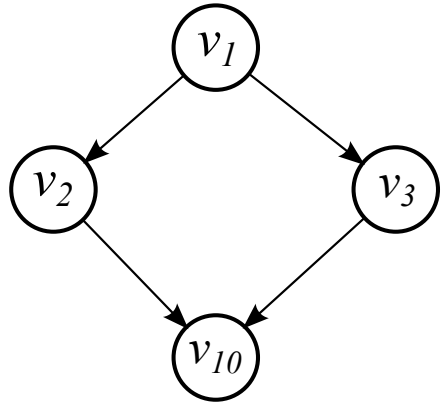


Example

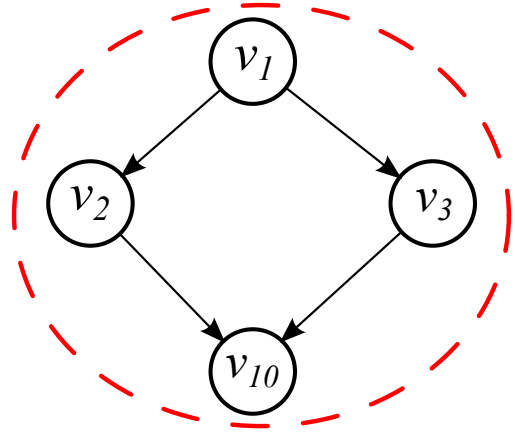


$$C(v_2) = 14||C(v_4)||C(v_5) = 1412161111$$

Example



Example



$$C(v_1) = 16||lex(C(v_2), C(v_3))||C(v_{10})$$
$$C(v_1) = 16131211141216111121$$

Example

$$C(G) = C(v_1) = 161312111412161111121$$

Correctness

Theorem 3 *Let G, G' de Dijkstra graphs, and $C(G), C(G')$ their encodings, respectively. Then G, G' are isomorphic if and only if $C(G) = C(G')$.*

Consequences

Corollary 2 *Let G be a DG and $C(G)$ its encoding.*

1. *There is a one-to-one correspondence between the 1's of $C(G)$ and the vertices of G.*
2. *The encoding $C(G)$ of G is unique and is a representation of G.*

Exhibiting the isomorphism function:

Corollary 3 *Let G, G' be DGs and $C(G), C(G')$ their corresponding encodings, satisfying $C(G) = C(G')$. Then an isomorphism function f between G and G' can be determined as follows. Let $v \in V(G)$ and $v' \in V(G')$ correspond to 1's at identical relative positions in $C(G)$ and $C(G')$, respectively. Define $f(v) := v'$.*

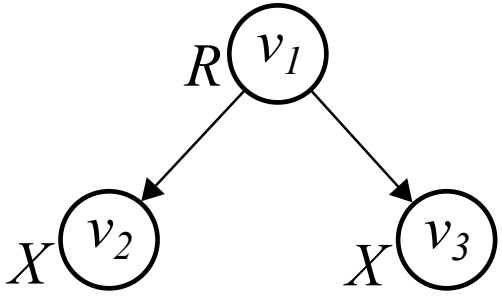
Complexity

Lemma 8 *Let G be a Dijkstra graph, and $C(G)$ its encoding; Then $|C(G)| \leq 2n - 1$.*

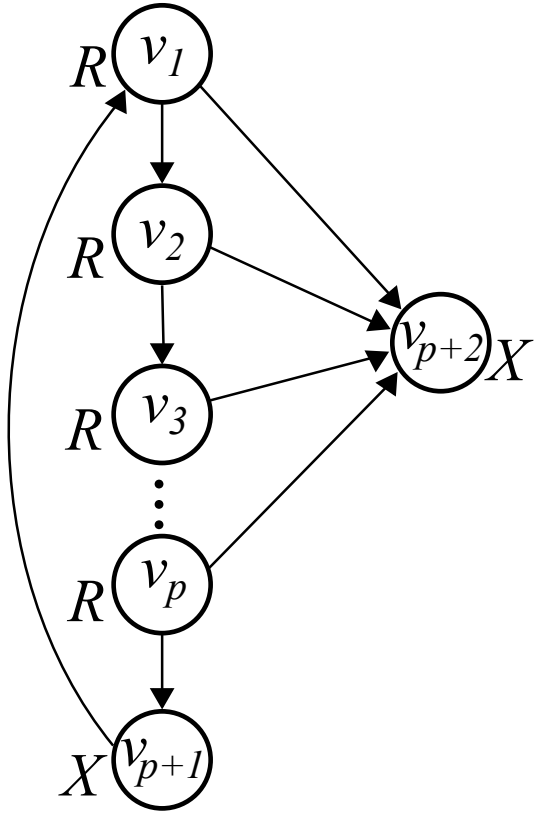
Theorem 4 *For DGs, the isomorphism algorithm terminates within $O(n)$ time.*

Generalization

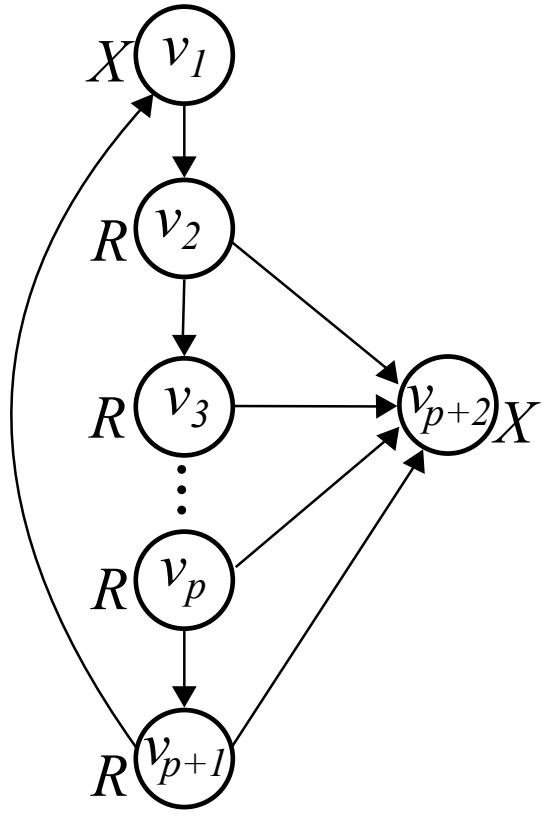
More useful statement graphs.



DIVERGENT IF THEN ELSE GRAPH

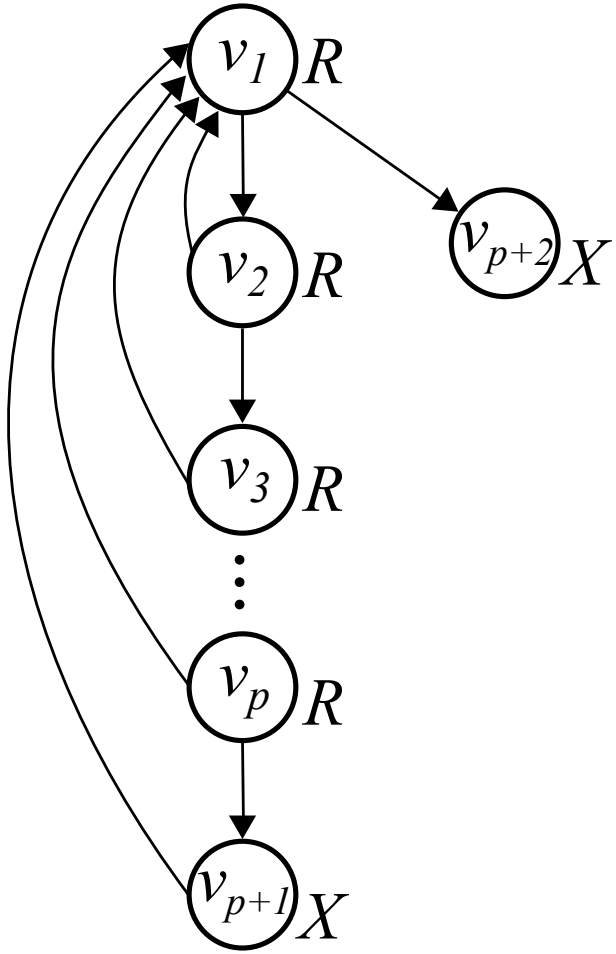


BREAK-WHILE GRAPH

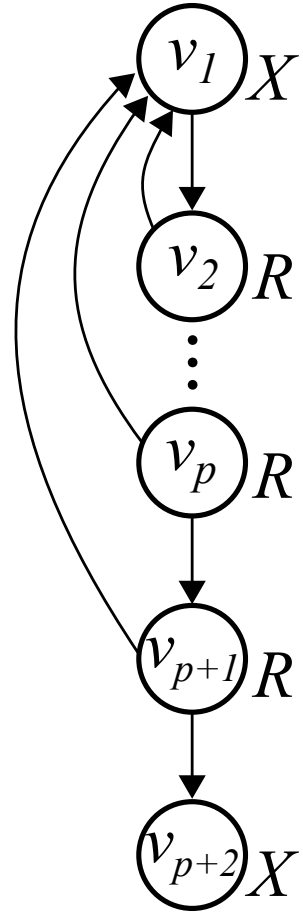


BREAK-REPEAT GRAPH

Generalization



CONTINUE-WHILE GRAPH



CONTINUE-REPEAT GRAPH

Applications

Recognition algorithm:
Graph watermarks

Isomorphism algorithm:

Code similarity analysis

THE END

THANK YOU

Edsger Wybe Dijkstra

(1930-2002)